Data Access 3

Integrating Hive and Kafka

Date of Publish: 2018-12-18

https://docs.hortonworks.com/

Contents

Apache Hive-Kafka integration

As a Hive user, you can connect to, analyze, and transform data in Kafka from Hive. You can offload data from Kafka to the Hive warehouse.

You connect to Kafka data from Hive by creating an external table that maps to a Kafka topic. The table definition includes a reference to a Kafka storage handler that makes the connection to Kafka. On the external table, Hive-Kafka integration supports ad hoc queries, such as questions about data changes in the stream within an interval of time just passed. You can transform Kafka data in the following ways:

- Perform data masking.
- Join dimension tables or any stream.
- Aggregate data.
- Change the Serde encoding of the original stream.
- Create a persistent stream in a Kafka topic.

You can achieve exactly once offloading of data by controlling its position in the stream. The Hive-Kafka connector supports the following serialization and deserialization formats:

- JsonSerDe (default)
- OpenCSVSerde
- AvroSerDe

Related Information

Apache Kafka Documentation

Create a table for a Kafka stream

You can create an external table in Hive that represents a Kafka stream to query real-time data in Kafka. You use a storage handler and table properties that map the Hive database to a Kafka topic and broker. If the Kafka data is not in JSON format, you alter the table to specify a serializer-deserializer for another format.

Procedure

1. Get the name of the Kafka topic you want to query to use as a table property.
   For example: "kafka.topic" = "wiki-hive-topic"
2. Construct the Kafka broker connection string.
   For example: "kafka.bootstrap.servers"="kafka.hostname.com:9092"
3. Create an external table named kafka_table using 'org.apache.hadoop.hive.kafka.KafkaStorageHandler' as shown in the following example:

```
CREATE EXTERNAL TABLE kafka_table
  (`timestamp` timestamp , `page` string,  `newPage` boolean,
  added int, deleted bigint, delta double)
  STORED BY 'org.apache.hadoop.hive.kafka.KafkaStorageHandler'
  TBLPROPERTIES
  ("kafka.topic" = "test-topic",
 "kafka.bootstrap.servers"="localhost:9092");
```

4. If the default JSON serializer-deserializer is incompatible with your data, choose another format in one of the following ways.

- Alter the table to use another supported serializer-deserializer. For example, if your data is in Avro format, use the Kafke serializer-deserializer for Avro:

```
ALTER TABLE kafka_table SET TBLPROPERTIES
 ("kafka.serde.class"="org.apache.hadoop.hive.serde2.avro.AvroSerDe");
```

- Create an external table that specifies the table in another format. For example, create a table named that specifies the Avro format in the table definition:

```
CREATE EXTERNAL TABLE kafka_t_avro
   (`timestamp` timestamp , `page` string,  `newPage` boolean,
   added int, deleted bigint, delta double)
   STORED BY 'org.apache.hadoop.hive.kafka.KafkaStorageHandler'
   TBLPROPERTIES
   ("kafka.topic" = "test-topic",
   "kafka.bootstrap.servers"="localhost:9092"
   -- STORE AS AVRO IN KAFKA
   "kafka.serde.class"="org.apache.hadoop.hive.serde2.avro.AvroSerDe");
```

Related Information

Apache Kafka Documentation

Querying live data from Kafka

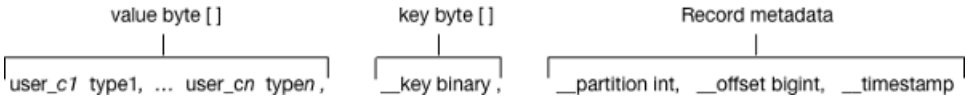
You can get useful information, including Kafka record metadata from a table of Kafka data by using typical Hive queries.

Each Kafka record consists of a user payload key (byte []) and value (byte[]) plus the following metadata fields:

- Partition int32
- Offset int64
- Timestamp int64

The Hive row represents the dual composition of Kafka data:

- The user payload serialized in the value byte array
- The metadata: key byte array, partition, offset, and timestamp fields



In the Hive representation of the Kafka record, the key byte array is called __key and is of type binary. You can cast __key at query time. Hive appends __key to the last column derived from value byte array, and appends the partition, offset, and timestamp to __key columns that are named accordingly.

Related Information

Apache Kafka Documentation

Query live data from Kafka

You can get useful information from a table of Kafka data by running typical queries, such as counting the number of records streamed within an interval of time and defining a view of streamed data over a period of time.

About this task

This task builds on the previous task in which you created a table named kafka_table for a Kafka stream.

Procedure

1. List the table properties and all the partition or offset information for the topic.

DESCRIBE EXTENDED kafka_table;

2. Count the number of Kafka records having timestamps within the last 10 minutes.

```
SELECT COUNT(*) FROM kafka_table
  WHERE `__timestamp` >  1000 * to_unix_timestamp(CURRENT_TIMESTAMP -
 interval '10' MINUTES);
```

Such a time-based seek requires Kafka 0.11 or later, which has a Kafka broker that supports time-based lookups; otherwise, this query leads to a full stream scan.

3. Define a view of data consumed within the last 15 minutes and mask specific columns.

```
CREATE VIEW last_15_minutes_of_kafka_table AS SELECT  `timestamp`, `user`,
 delta,
  ADDED FROM kafka_table
  WHERE `__timestamp` >  1000 * to_unix_timestamp(CURRENT_TIMESTAMP -
 interval '15' MINUTES) ;
```

4. Create a dimension table.

```
CREATE TABLE user_table (`user` string, `first_name` string , age int,
 gender string, comments string) STORED as ORC ;
```

5. Join the view of the stream over the last 15 minutes to the user_table, group by gender, and compute aggregates over metrics from fact table and dimension tables.

```
SELECT SUM(added) AS added, SUM(deleted) AS deleted, AVG(delta) AS delta,
 AVG(age) AS avg_age , gender
  FROM last_15_minutes_of_kafka_table
  JOIN user_table ON `last_15_minutes_of_kafka_table`.`user` =
 `user_table`.`user`
  GROUP BY gender LIMIT 10;
```

6. Perform a classical user retention analysis over the Kafka stream that does a stream-to-stream join to run adhoc queries on a view defined over the past 15 minutes.

```
-- Stream join over the view itself
-- Assuming l15min_wiki is a view of the last 15 minutes
SELECT  COUNT( DISTINCT activity.`user`) AS active_users,
COUNT(DISTINCT future_activity.`user`) AS retained_users
FROM l15min_wiki AS activity
LEFT JOIN l15min_wiki AS future_activity ON activity.`user` =
 future_activity.`user`
AND activity.`timestamp` = future_activity.`timestamp` - interval '5'
 minutes ;

--  Stream-to-stream join
-- Assuming wiki_kafka_hive is the entire stream.
SELECT floor_hour(activity.`timestamp`), COUNT( DISTINCT activity.`user`)
 AS active_users,
COUNT(DISTINCT future_activity.`user`) as retained_users
FROM wiki_kafka_hive AS activity
LEFT JOIN wiki_kafka_hive AS future_activity ON activity.`user` =
 future_activity.`user`
AND activity.`timestamp` = future_activity.`timestamp` - interval '1'
 hour
GROUP BY floor_hour(activity.`timestamp`);
```

Related Information

Apache Kafka Documentation

Write transformed Hive data to Kafka

Perform ETL: Ingest data from Kafka into Hive

You can extract, transform, and load a Kafka record into Hive in a single transaction.

Procedure

1. Create a table to represent source Kafka record offsets.

```
CREATE TABLE kafka_table_offsets(partition_id int, max_offset bigint,
 insert_time timestamp);
```

2. Initialize the table.

```
INSERT OVERWRITE TABLE kafka_table_offsets
SELECT `__partition`, min(`__offset`) - 1, CURRENT_TIMESTAMP
FROM wiki_kafka_hive
GROUP BY `__partition`, CURRENT_TIMESTAMP;
```

3. Create the destination table.

```
CREATE TABLE orc_kafka_table (partition_id int, koffset bigint, ktimestamp
 bigint,
  `timestamp` timestamp , `page` string, `user` string, `diffurl` string,
  `isrobot` boolean, added int, deleted int, delta bigint
) STORED AS ORC;
```

4. Insert Kafka data into the ORC table.

```
FROM wiki_kafka_hive ktable JOIN kafka_table_offsets offset_table
ON (ktable.`__partition` = offset_table.partition_id
AND ktable.`__offset` > offset_table.max_offset )
INSERT INTO TABLE orc_kafka_table
SELECT `__partition`, `__offset`, `__timestamp`,
  `timestamp`, `page`, `user`, `diffurl`, `isrobot`, added , deleted ,
 delta
INSERT OVERWRITE TABLE kafka_table_offsets
SELECT `__partition`, max(`__offset`), CURRENT_TIMESTAMP
GROUP BY `__partition`, CURRENT_TIMESTAMP;
```

5. Check the insertion.

```
SELECT MAX(`koffset`) FROM orc_kafka_table LIMIT 10;

SELECT COUNT(*) AS c FROM orc_kafka_table
GROUP BY partition_id, koffset HAVING c > 1;
```

6. Repeat step 4 periodically until all the data is loaded into Hive.

Writing transformed data to Kafka

You can extract, transform, and load a Hive table to a Kafka topic for real-time streaming of a large volume of Hive data. You need some understanding of write semantics and the metadata columns required for writing data to Kafka.

Write semantics

The Hive-Kafka connector supports the following write semantics:

- At least once (default)
- Exactly once

At least once (default)	At least once is the most common write semantic used by streaming engines. The internal Kafka producer retries on errors. In the event of an undelivered message, the exception is raised to the task level that causes its restart, and thus more retries. At least once leads to one of the following conclusions: • If the job succeeds, each record is guaranteed to be delivered at least once. • If the job fails, some of the records might be lost and some might not be sent. You can retry the query, which eventually leads to the delivery of each record at least once.
Exactly once	Following the exactly once semantic, the Hive job ensures that either every record is delivered exactly once, or nothing is delivered. You can use only Kafka brokers supporting the Transaction API (0.11.0.X or later). To use this semantic, you need to set the following table property "kafka.write.semantic"="EXACTLY_ONCE".

Metadata columns

In addition to the user row payload, the insert statement must include values for the following extra columns:

__key	You can set the value of this metadata column to null, but using a meaningful key value to avoid unbalanced partitions is recommended. Any binary value is valid.
__partition	The recommended value is null unless you want to route the record to a particular partition. Do not use a non-existing partition value. Doing so results in an error.
__offset	The value is fixed at -1. Kafka does not allow you to set this value.
__timestamp	You can set this value to a meaningful timestamp, represented as the number of milliseconds since epoch. Optionally, you can set this value to null or -1, which means the Kafka broker strategy sets the timestamp column.

Related Information

Apache Kafka Documentation

Write transformed Hive data to Kafka

You can extract a Kafka input topic, transform the record in Hive, and load a Hive table back into a Kafka record.

About this task

In this task, you build upon the previous task, “Query live data from Kafka,” by reading the Kafka input topic named l15min_wiki. When you transform the record in the Hive execution engine, you compute a moving average over a window of one minute. The resulting record that you write back to another Kafka topic is named moving_avg_wiki_kafka_hive.

.

Procedure

1. Create an external table to represent the Hive data you want to load into Kafka.

```
CREATE EXTERNAL TABLE moving_avg_wiki_kafka_hive
(`channel` string, `namespace` string,`page` string, `timestamp`
 timestamp , avg_delta double )
STORED BY 'org.apache.hadoop.hive.kafka.KafkaStorageHandler'
TBLPROPERTIES
  ("kafka.topic" = "moving_avg_wiki_kafka_hive",
  "kafka.bootstrap.servers"="kafka.hostname.com:9092",
  -- STORE AS AVRO IN KAFKA
  "kafka.serde.class"="org.apache.hadoop.hive.serde2.avro.AvroSerDe");
```

2. Insert data, whch you select from the Kafka topic, back into the Kafka record.

```
INSERT INTO TABLE moving_avg_wiki_kafka_hive
SELECT `channel`, `namespace`, `page`, `timestamp`,
  AVG(delta) OVER (ORDER BY `timestamp` ASC ROWS BETWEEN  60 PRECEDING AND
 CURRENT ROW) AS avg_delta,
  null AS `__key`, null AS `__partition`, -1 AS `__offset`, to_epoch_milli
(CURRENT_TIMESTAMP) AS `__timestamp`
FROM l15min_wiki;
```

   The timestamps of the selected data are converted to milliseconds since epoch for clarity.

Related Information

Query live data from Kafka

Set consumer and producer properties as table properties

You can use Kafka consumer and producer properties in the TBLPROPERTIES clause of a Hive query. By prefixing the key with kafka.consumer or kafka.producer, you can set the table properties.

Procedure

Inject 5000 poll records into the Kafka consumer.

```
ALTER TABLE kafka_table SET TBLPROPERTIES
 ("kafka.consumer.max.poll.records"="5000");
```

Kafka storage handler and table properties

You use the Kafka storage handler and table properties to specify the query connection and configuration.

Kafka storage handler

You specify 'org.apache.hadoop.hive.kafka.KafkaStorageHandler' in queries to connect to, and transform a Kafka topic into, a Hive table. In the definition of an external table, the storage handler creates a view over a single Kafka topic. For example, to use the storage handler to connect to a topic, the following table definition specifies the storage handler and required table properties: the topic name and broker connection string.

```
CREATE EXTERNAL TABLE kafka_table
   (`timestamp` timestamp , `page` string,  `newPage` boolean,
   added int, deleted bigint, delta double)
   STORED BY 'org.apache.hadoop.hive.kafka.KafkaStorageHandler'
   TBLPROPERTIES
```

```
   ("kafka.topic" = "test-topic",
 "kafka.bootstrap.servers"="localhost:9092");
```

kafka.topic	The Kafka topic to connect to
kafka.bootstrap.servers	The broker connection string

Storage handler-based optimizations

The storage handler can optimize reads using a filter push-down when executing a query such as the following time-based lookup supported on Kafka 0.11 or later. The Kafka consumer supports seeking on the stream based on an offset, which the storage handler leverages to push down filters over metadata columns.

```
SELECT COUNT(*) FROM kafka_table
   WHERE `__timestamp` >  1000 * to_unix_timestamp(CURRENT_TIMESTAMP -
 interval '10' MINUTES) ;
```

The storage handler in this example performs seeks based on the Kafka record __timestamp to read only recently arrived data.

The following logical operators and predicate operators are supported in the WHERE clause:

Logical operators: OR, AND

Predicate operators: <, <=, >=, >, =

The storage handler reader optimizes seeks by performing partition pruning to go directly to a particular partition offset used in the WHERE clause.

```
SELECT COUNT(*)  FROM kafka_table
  WHERE (`__offset` < 10 AND `__offset` > 3 AND `__partition` = 0)
  OR (`__partition` = 0 AND `__offset` < 105 AND `__offset` > 99)
  OR (`__offset` = 109);
```

The storage handler scans partition 0 only, and then read only records between offset 4 and 109.

Kafka metadata

In addition to the user-defined payload schema, the Kafka storage handler appends to the table the following additional columns, which you can use to query the Kafka metadata fields:

__key	Kafka record key (byte array)
__partition	Kafka record partition identifier (int 32)
__offset	Kafka record offset (int 64)
__timestamp	Kafka record timestamp (int 64)

The partition identifier, record offset, and record timestamp plus a key-value pair constitute a Kafka record. Because the key-value is a 2-byte array, you need to use a Serde classes to transform the array into a set of columns.

Table Properties

You can use the following properties in the TBLPROPERTIES clause of a Hive query that specifies the Kafka storage handler:

Property	Description	Required	Default
kafka.topic	Kafka topic name to map the table to.	Yes	null

Property	Description	Required	Default
kafka.bootstrap.servers	Table property indicating Kafka broker(s) connection string.	Yes	null
kafka.serde.class	Serializer and Deserializer class implementation.	No	org.apache.hadoop.hive.serde2.JsonSerDe
hive.kafka.poll.timeout.ms	Parameter indicating Kafka Consumer poll timeout period in millis. FYI this is independent from internal Kafka consumer timeouts.	No	5000 (5 Seconds)
hive.kafka.max.retries	Number of retries for Kafka metadata fetch operations.	No	6
hive.kafka.metadata.poll.timeout.ms	Number of milliseconds before consumer timeout on fetching Kafka metadata.	No	30000 (30 Seconds)
kafka.write.semantic	Writer semantic, allowed values (NONE, AT_LEAST_ONCE, EXACTLY_ONCE)	No	AT_LEAST_ONCE